A Weighted Symmetric Graph Embedding Approach for Link Prediction in Undirected Graphs

Zhixiao Wang[a,b], Yahui Chai[a], Chengcheng Sun[a], Xiaobin Rui[a,b,*], Hao Mi[a], Youran Pan[c], Xinyu Zhang[d,*]

[a]*School of Computer Science, China University of Mining and Technology, Xuzhou Jiangsu, 221116, China*
[b]*Mine Digitization Engineering Research Center of the Ministry of Education, Xuzhou Jiangsu, 221116, China*
[c]*Duke Kunshan University, Kunshan Jiangsu, 215316, China*
[d]*School of Computer Science, Hunan University of Technology and Business, Changsha Hunan, 410205, China*

Abstract

Link prediction is an important task in social network analysis and mining because of its various applications. A large number of link prediction methods have been proposed. Among them, the deep learning-based embedding method exhibits excellent performance, which encodes each node and edge as an embedding vector, enabling easy integration with traditional machine learning algorithms. However, there still remain some unsolved problems for this kind of methods, especially in the steps of node embedding and edge embedding. Firstly, they share the same weight, rather than assign different reasonable weights, to all neighbors of different orders to obtain the node embedding. Secondly, they can hardly keep the symmetry of edge embeddings obtained from node representations by direct concatenation or other binary operations like mean and Hadamard Product. In order to solve these problems, we propose a weighted symmetric graph embedding approach for link prediction. In node embedding, the proposed approach aggregates neighbors in different orders with different weights. Specifically, the weight of the k^{th}-order neighbors is set to $1/\langle d\rangle^{k-1}$ where $\langle d\rangle$ is the average degree of the network. In edge embedding, the pro-

*Corresponding author
Email addresses: ruixiaobin@cumt.edu.cn (Xiaobin Rui), zhangxinyu247@163.com (Xinyu Zhang)

posed approach bi-directionally concatenates node pairs both forwardly and backwardly to guarantee the symmetry of edge representations while preserving local structural information. Experimental results show that our proposed approach can better predict links in a network, outperforming the state-of-the-art methods. The appropriate weight assignment and the bi-directional concatenation enable us to learn more accurate and symmetric edge representations for link prediction.

Keywords: graph embedding, graph neural network, link prediction, weighted aggregation, symmetric concatenation

1. Introduction

A social network reflects the reality of human communities, in such networks, a node refers to a person or social entity, and a link corresponds to the relationships between them. Social networks are highly dynamic and complex because of the continuously changing of these relationships. Link prediction is an important task in social network analysis and mining, which aims to predict missing links or links that are likely to occur in the future [1, 2]. There are numerous applications related to link prediction, such as recommendation systems [3], knowledge graph completion [4], influence analysis [5], community detection [6], even face clustering [7] and multiple object tracking [8].

In recent decades, a large number of link prediction methods have been proposed, which can be grouped into three categories, i.e. similarity-based [9], learning-based [10] and embedding-based methods [11]. A couple of survey literatures [9, 10, 11, 12] reviewed the existing link prediction methodologies and provided comprehensive analysis from different concerns.

Similarity-based methods [9, 10, 11] calculate the similarity of node pair based on the shared local, global or hybrid structural properties, including node neighborhoods, paths between nodes, etc. A similarity score is assigned to each unobserved link and the top-k links with the highest score are predicted. Although simple and working well in practice, this kind of methods is based

on an assumption that two nodes are more likely to connect if they have many common neighbors, which limits their effectiveness on networks where these assumptions fail [13]. Considering this regard, a more reasonable way should be learning suitable heuristic features from a given network instead of predefining [13].

Learning-based methods [10, 11, 12] use a model to learn with given features, extract latent patterns and eventually predict potential links. Available models include the probabilistic model [14], traditional supervised learning model [15], and deep model [16]. Different from similarity-based methods, learning-based methods focus on not only the topological information but also the non-topological information of social networks [10], and all the information is used as an entry of a feature vector for learning a model. However, extracting some non-topological information is not easy, the curse of dimensionality is another challenge for learning-based methods [11].

In order to solve the dimensional disaster problem, embedding-based methods [11] map the higher-dimensional nodes of a graph to a lower-dimensional vector space by preserving the node neighborhood structures and other network properties. Recently, this kind of methods has become widely popular due to its own advantages in capturing inherent dynamics of the network either explicitly or implicitly [17]. In addition, Grover et al. [18] found that embedding is more accurate than traditional similarity-based or learning-based link prediction methods. Embedding technique mainly includes three categories [17]: matrix factorization [19, 20, 21] , random walk [22, 18] and deep learning [23, 24]. The growing research on deep learning based embedding has led to a deluge of applications to link prediction. Generally speaking, this kind of methods first obtains the node representation and then extracts the edge representation by combining the node representation to predict the likely but unobserved links finally. Goyal et al. [17] demonstrated the excellent performance of deep learning based embedding in the task of link prediction. However, there still remain some unsolved problems for this kind of methods:

(1) When obtaining the node representation, many traditional methods take

not only the first-order neighbors but also some high-order neighborhood information of the target nodes into account to guarantee the accuracy of node embedding. However, as far as we know, they all ignore the difference between different orders. In fact, although the target node in social networks interacts not only with first-order neighbors but also with higher-order neighbors, the contributions of different orders are different. Obviously, the first-order neighbors are more important than the second-order in node embedding. Therefore, they should be assigned with different weights.

(2) When obtaining the edge representation from node representation, some methods directly concatenate node representations [3], and others mix node representations by averaging or through Hardmard product [4]. The first branch could preserve local structural information but can hardly keep the symmetry of embedded edges. Suppose there are two nodes u and v in an undirected graph. Apparently, the edge embedding of (u, v) and (v, u) should be the same. The second branch could guarantee symmetry but at a cost of losing some local structural information. In summary, neither of the above two methods could obtain a high-qualified edge embedding that balances both local structural information and symmetric presentation, which will affect the performance of link prediction.

To fill the above two gaps, this paper proposes a novel embedding approach for link prediction named Weighted Symmetric Graph Embedding (WSGE). The proposed approach first assigns weights to different orders of neighbors to construct the weight matrix. Then, the propagation and updating of a GNN (Graph Neural Network) are completed based on the weight matrix to obtain the embedded node representation. After that, node pairs are concatenated bidirectionally (both forwardly and backwardly) and then mixed to guarantee the symmetry of edge representations while preserving local structural information. The embedded edges serve as the input of DNN (Deep Neural Networks) to predict the links finally.

The contributions of this paper are summarized as follows.

(1) To solve the problem that all orders of neighbors are viewed equally in

traditional node embedding, this paper proposes a novel method to aggregate neighbors in different orders with different weights. Specifically, the weight matrix is constructed by aggregating the adjacency matrices with different powers corresponding to different weights. The weight of the k^{th}-order neighbors is set to $1/\langle d\rangle^{k-1}$ where $\langle d\rangle$ is the average degree of the network. On one hand, the weights that balance multi-order neighbors can improve the propagating and updating process of GNN to obtain node representations. On the other hand, due to the k^{th} row in the weight matrix contains all the weights of multi-order neighbors for updating the k^{th} node, our method could use mini-batch forward propagation to train the GNN while other methods have to use the whole graph data as the input.

(2) To solve the problem that traditional edge embedding methods can hardly keep both local structural information and symmetric presentation simultaneously, this paper proposes a novel approach to combine node representations effectively. Specifically, node pairs are first concatenated bi-directionally with full local information to obtain two opposite edges. Then, these edges are trained separately through a DNN to obtain their corresponding representations. After that, each pair of trained opposite edge representations are combined through a mixing operation to guarantee the symmetry of the final embedded edges.

Experimental results on five real-world networks demonstrate that the proposed WSGE significantly improves the state-of-the-art by learning more accurate and symmetric edge representations that better preserve the graph structure. Thus, we can better complete the link prediction task in undirected graphs.

2. Related Works

Traditional embedding-based link prediction methods mainly include three categories, i.e. matrix factorization [19, 20, 21], random walk [22, 18], and deep learning [23, 24].

The matrix factorization [19] is a typical dimensionality reduction technique for link prediction, which represents the connections between nodes in the form

of a matrix and factorizes this matrix to obtain the node embedding. Compared with previous methods, the node embedding obtained by matrix factorization could improve the efficiency of link prediction. Local Linear Embedding (LLE) [21] is a classic matrix factorization method for node embedding which assumes that each node is a linear combination of its neighbors in the embedding space. Followed by LLE, M. Belkin et al. [20] proposed Laplacian Eigenmaps (LE) to obtain the node embedding by decomposing the Laplacian matrix. Similarly, Ahmed A et al. [25] proposed Graph Factorization (GF) to obtain the node embedding by decomposing the adjacency matrix. Due to that these two methods only involve first-order neighbors, they would fail to capture the global information of the graph. GrapRep [26] extends LE and GF to preserve high-order proximity by decomposing the k-order of transition probability matrix rather than only the first order. Though GrapRep takes into account the k-order neighbors, it directly integrates features of each order without considering the weights of different orders. Besides, Wang et al. [27] proposed the joint embedding method which considers the set of graphs together. It takes a matrix factorization approach to extract features for multiple graphs.

Random walk methods including DeepWalk [22] and Node2vec [18]. They are both based on a neural language model, i.e. SkipGram [28], which aims to maximize the co-occurrence probability among words that appear within a window . DeepWalk first generates a large number of random walk sequences through sampling. Then it uses SkipGram and hierarchical softmax to model the probability of the node pair in each local window. After that, it maximizes the co-occurrence probability among nodes that appear within a window to obtain the context information (of nodes) for node embedding. Node2vec further generalizes DeepWalk with Breadth-First Search (BFS) and Depth-First Search (DFS) on random walks. It also employs SkipGram with negative sampling to learn the node representation [29]. These methods involve high-order neighbors when applying random walks, which could produce node representations that preserve the graph structure [30]. However, the nodes in the same window share a uniform weight, ignoring the weight distribution of each node's neighborhood.

As a deep learning model, the convolutional neural network (CNN) has been widely used in graph embedding, thus graph neural networks (GNNs) [31] emerged. GNNs combine the feature information and the graph structure to learn better representations of graphs via feature propagation and aggregation [32]. In recent years, embedding techniques that exploit GNNs to obtain node representations have been proposed, such as GCN [33], GraphSAGE [34], and GAT [35]. GCN is a scalable model since it only aggregates features from local neighborhoods. Meanwhile, the embedded results of GCN could also characterize global neighborhoods through multiple iterations. GraphSAGE generates embeddings by sampling the local neighborhood for feature aggregation. In this way, it can use a mini-batch forward propagation algorithm to train data. GAT, based on the spatial graph convolution network, introduces attention mechanism into propagation process, thus the hidden state of each node is calculated by paying attention to neighbors. However, GAT only involves first-order neighbors and assigns different weights to all these neighbors, resulting in higher complexity.

All above embedding methods are node-based, and the edges are mapped to low-dimensional vectors indirectly. More recently, an edge-based embedding method Edge2vec [36] is proposed, which maps the edges to a low-dimensional space directly. Meanwhile, a method of co-embedding nodes and edges [37] is also proposed, which directly embeds both nodes and edges to a latent feature space. Both of them are effective methods for graph embedding. However, the number of edges is usually far greater than that of nodes in a network, thus embedding edges directly may cost more time [36], which makes it inefficient for link prediction.

To sum up, when embedding nodes, some methods view all orders of neighbors equally, ignoring the difference between them. While other methods assign different weights to all neighbors, resulting in higher complexity. Therefore, this paper proposes a novel approach to aggregate neighbors in different orders with different weights. The weights that balance multi-order neighbors can improve the propagating and updating process of GNN to obtain node rep-

resentations. Moreover, traditional edge embedding methods including directly concatenating or Hadamard Product can hardly keep both local structural information and symmetric presentation simultaneously. Hence, this paper further proposes a novel concatenating approach where node pairs are concatenated bi-directionally with full local information to obtain two opposite edges. Then, each pair of trained opposite edge representations are combined through a mixing operation to guarantee the symmetry of the final embedded edges.

3. Method

To better solve the link prediction problem, we propose a novel Weighted Symmetric Graph Embedding (WSGE) approach, which consists of four parts: one-hot encoding, weighted node embedding, symmetric edge embedding, and link prediction. The framework of WSGE is shown in Figure 1.

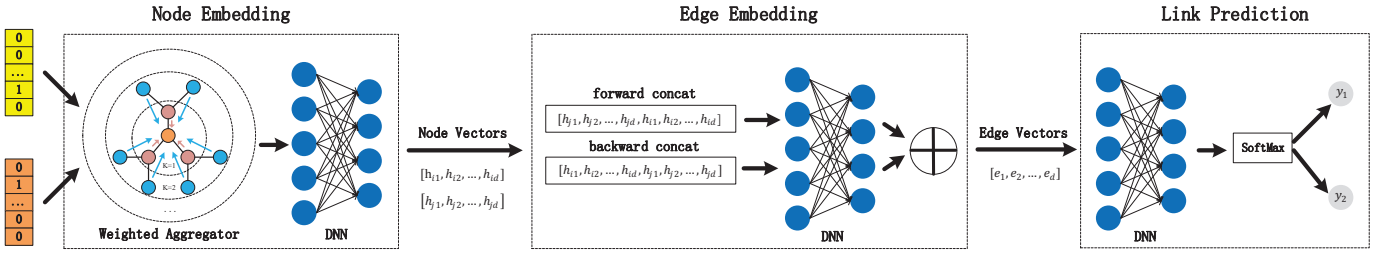


Figure 1: Framework of the Weighted Symmetric Graph Embedding (WSGE) method.

Firstly, WSGE obtains the one-hot encoding vector of each node. Secondly, it extracts the node embedding by aggregating and updating node features based on multi-order neighbors with different weights. The final node embeddings are obtained through feature space mapping which is implemented by a multi-layer perceptron (MLP). Thirdly, edge embeddings are acquired based on bi-directional concatenating to ensure the uniqueness of edges as well as preserving the local structure. Finally, with the edge embedding results, the link prediction problem is transformed into a binary classification task and a fully connected

neural network is employed to predict whether there is a link between two nodes.

3.1. One-hot encoding

The initial nodes have no features, which makes it impossible to reveal the differences between nodes through aggregating. Thus, WSGE applies one-hot encoding to make the features of each node different from each other. For a network with N nodes, the one-hot encoding of node i is an N-dimensional vector(1*N) with the k^{th} element being 1 and all others 0. The link prediction task aims to predict missing links or links that are likely to occur between two nodes. Therefore, the inputs are the vectors of two nodes encoded by one-hot encoding, respectively.

3.2. Weighted node embedding

A node in social networks interacts with not only first-order neighbors but also higher-order neighbors. Moreover, aggregating different order neighbors with the same weight will make the embeddings more biased towards the high-order neighbors and causes the loss of local structural information. Therefore, we should not only take multi-order neighbors into account but also assign them with different weights during the node aggregating process.

According to general GNNs, node embedding must involve two steps, i.e. neighbor features propagation and target node status update. In order to avoid the node iteration and accelerate the training process, our proposed WSGE approach integrates those two steps into a weighted matrix, which is defined as follows:

$$\overline{A} = \sum_{k} \delta_k A^k \tag{1}$$

where k is the order of neighbors. A^k is a k-hop adjacency matrix that reflects the strength of neighborhood relationship in k hops between two nodes, which is used to preserve the high-order neighbors' features. δ_k is the weight for k^{th}-order neighbors.

As previously mentioned, aggregating different order neighbors with the same weight will make the embeddings more biased towards the high-order neighbors. Therefore, to balance neighbors in different orders, we set δ_k to $1/\langle d\rangle^{k-1}$ where $\langle d\rangle$ is the average degree of the corresponding network. The average degree implies the exponential relationship w.r.t. the number of nodes in different orders. By introducing $1/\langle d\rangle^{k-1}$, a kind of quantity-aware weights could be assigned to multi-order neighbors when aggregating them to the central node. Consequently, different order neighbors would be roughly balanced according to their quantities.

Then, based on $\overline{A}$, the weighted vector of each node is represented as

$$a_i = x_i \overline{A} \tag{2}$$

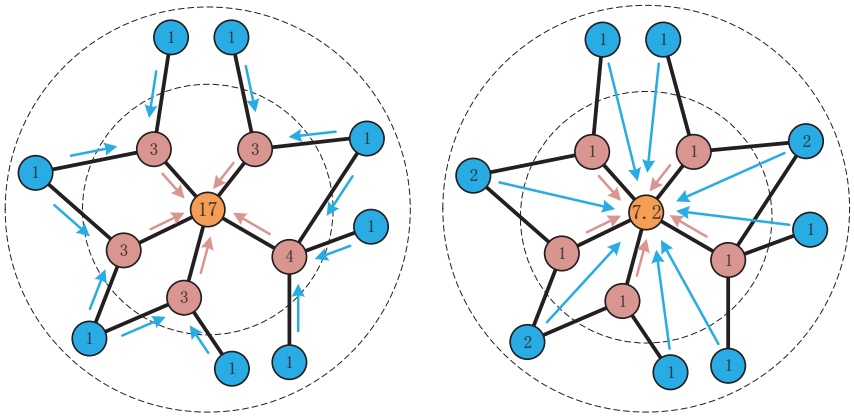
where x_i is the one-hot encoding vector of node i. Through multiplying the one-hot encoding vector x_i with a weighted matrix $\overline{A}$, a_i represents the i^{th} row in the weighted matrix which contains all the weights of multi-order neighbors for updating the i^{th} node. In this manner, our method could use mini-batch forward propagation to train the GNN while other methods have to use the whole graph data as the input.

WSGE does not perform any training on the initial embedding state but retains the local structure of the graph through aggregating, updating, and feature space mapping. Therefore, the initial node embeddings h_i^0 can be obtained by the following formula:

$$h_i^0 = a_i H^0 \tag{3}$$

where H^0 is the whole graph embedding, which is obtained based on random initialization of normal distribution. Through matrix multiplication of a_i and H^0, the initial node embedding h_i^0 can be acquired. Meanwhile, the two steps of the neighbor feature propagation and target node status updating can be completed, Thus, WSGE reduces the iterative process of aggregation and update in the general GNN model.

In the following, we use a schematic network (shown in Figure 2) to show the difference between the traditional node aggregation method and the weighted aggregation method proposed in this paper. The representations of all nodes are initialized as 1 rather than vectors for better observing the aggregating process.



(a) traditional aggregation (b) weighted aggregation

Figure 2: Different node aggregation methods.

Figure 2(a) shows the process of the traditional node aggregation. The target node has five first-order neighbors and eight second-order neighbors. The five first-order neighbors need to aggregate their own neighborhood (the second-order neighbors of the target node) to obtain corresponding embeddings, and then aggregate their embeddings to the target node. Finally, the target node is represented as 17 according to the embedding result. In contrast, our proposed approach does not need any iteration. As shown in Figure 2(b), through a customized weight matrix, multi-order neighbors can be directly aggregated to the target node and then the embedding of the target node is obtained. The target node is represented as 7.2 according to the embedding result, which is closer to the aggregation value 6 of involving only the first-order neighbors. In other words, our proposed weighted aggregation can ensure that the first-order neighbors have a greater impact on the target node than the second-order. Thus, the embedding result would not be overly covered by high-order

neighbors, which could well retain the local structural information. Moreover, multi-order neighbors are directly aggregated to the target node, which enables the mini-batch forward propagation for training the GNN.

After neighbor features propagation and target node status updating, a multi-channel single-layer neural network is adopted to map the obtained feature vector to a low-dimensional space. Formally, the mapping process is represented as

$$\overline{h_i} = \sigma(W_1 h_i^0 + b_1) + \sigma(W_2 h_i^0 + b_2) \tag{4}$$

where $\overline{h_i}$ is the final node embedding, W_1, W_2 are the weight matrices, b_1, b_2 are the bias in the fully connected neural network. σ is the activation function. Through feature space mapping, the final node embedding $\overline{h_i}$ can be obtained.

In summary, WSGE employs a weighted matrix to obtain node embeddings by completing neighbor feature propagation and node status update, which not only balances neighbors in different orders through appropriate weight assignment but also enables mini-batch forward propagation for training.

3.3. Symmetric edge embedding

Extracting the edge embedding from the node embedding is a key step for link prediction. During this process, the following two conditions should be met: 1) For a pair of nodes i, j, the embeddings of (i, j) and (j, i) should be the same in an undirected graph. 2) The edge embeddings generated from node representations should retain the local structural features of both the edge and the corresponding node pair.

However, some traditional methods directly concatenate node embeddings [3, 36, 37] and others mix node embeddings by averaging or Hadamard Product [4, 30]. The former will cause the discrepancy between (i, j) and (j, i), i.e. there will be two different representations for the same edge. Take the schematic network shown in Figure 3(a) as an example, the node embeddings of two nodes in orange are 4 and 2, respectively, the direct concatenating method will produce two different edge representations, i.e. (4,2) and (2,4). The latter can obtain the same edge representation even from different node representations, however,

it will result in the loss of local structural information. Take the schematic network shown in Figure 3(b) as another example, the node embeddings of two nodes in orange are both 3. If the averaging or Hadamard Product methods are employed to generate the edge embedding, the results of the two nodes in Figure 3(a) would be the same as the results of the two nodes in Figure 3(b), failing to retain the local structural information of these schematic networks. In summary, none of those methods could obtain a high-qualified edge embedding that balances both local structural information and symmetric presentation, which will affect the performance of link prediction.

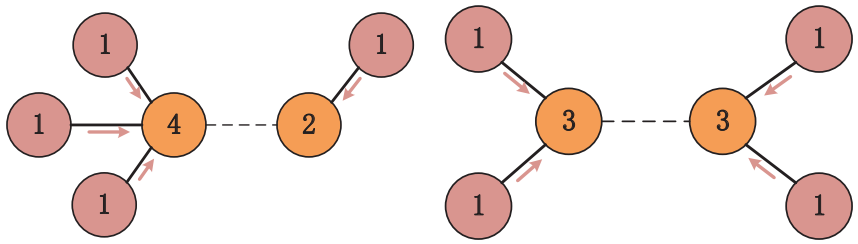


Figure 3: Traditional concatenating methods.

WSGE provides a bi-directional concatenating approach to generate the corresponding edge representations, which concatenates node pair's representations bi-directionally. Suppose there are two nodes i,j, and the final node embeddings $\overline{h_i}$, $\overline{h_j}$ are d-dimensional vectors. Firstly, the node embeddings $\overline{h_i}$, $\overline{h_j}$ will be concatenated bi-directionally to compose two $2d$-dimensional edge embedding vectors $g_{forward}(\overline{h_i}, \overline{h_j})$, $g_{backward}(\overline{h_i}, \overline{h_j})$. Secondly, these two vectors are put into a deep neural network to generate two d-dimensional edge embeddings $f(g_{forward}(\overline{h_i}, \overline{h_j}))$, $f(g_{backward}(\overline{h_i}, \overline{h_j}))$. Finally, we can obtain the final embedded edge representation $e_{i,j}$ from these two d-dimensional edge embeddings with an addition operation. Specifically, the final edge embedding can be defined as follows:

$$e_{i,j} = f(g_{forward}(\overline{h_i}, \overline{h_j})) + f(g_{backward}(\overline{h_i}, \overline{h_j})) \tag{5}$$

where $e_{i,j}$ is the final edge embedding. $g_{forward}(\overline{h_i}, \overline{h_j})$ and $g_{backward}(\overline{h_i}, \overline{h_j})$ denotes the representations of concatenating $\overline{h_i}$, $\overline{h_j}$ and $\overline{h_j}$, $\overline{h_i}$ respectively. f represents a deep neural network that maps a $2d$-dimensional edge embedding vector into d-dimension.

The structure of the deep neural network is shown in Figure 4, which contains two fully connected layers. Following each fully connected layer, there is a one-dimensional batch normalization layer to ensure that the input of each layer is equally distributed during the training process. $Relu(x) = max(0, x)$ is an element-wise activation function.

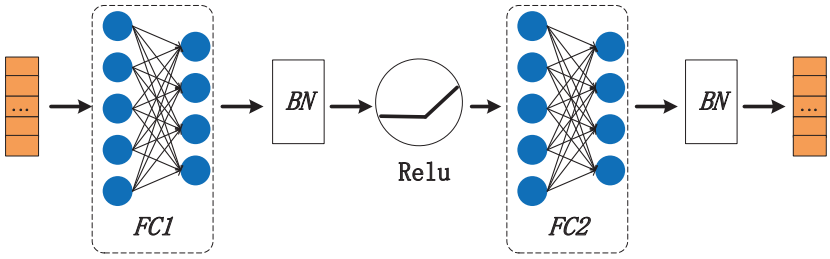


Figure 4: The structure of the deep neural network.

In summary, our proposed concatenating approach can ensure the identity between edge embeddings $e_{i,j}$ and $e_{j,i}$ while preserving the local structural features of the graph, avoiding the problem of lacking either symmetric presentation or structural information in traditional concatenating methods.

3.4. Link prediction

Our proposed WSGE obtains the node embeddings by aggregating nodes of different orders with different weights, then these embedding results are concatenated bi-directionally to obtain the edge embeddings. After that, the link prediction problem is transformed into a binary classification. WSGE employs a fully connected neural network to this task, and Figure 5 shows the details of the neural network.

Since the dimension of the edge embedding is d, the input dimension of the fully connected network is also d. Due to that the output has only two cate-

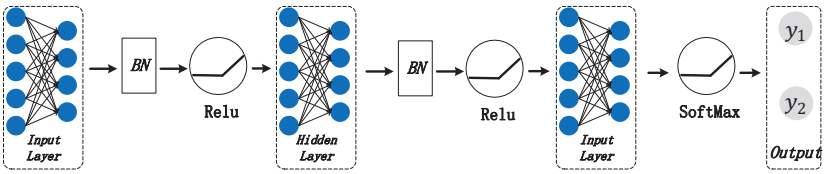


Figure 5: Link prediction process.

gories, the output dimension is set to 2. For the stability of the entire training process, the input layer and hidden layer are followed by a batch normalization layer, and Relu is adopted as the activation function. The output layer utilizes Softmax as the activation function. Thus, the final link prediction result is a two-dimensional vector $[y_1, y_2]$.

3.5. Algorithm description

The pseudocode of the proposed WSGE approach is shown in Algorithm 1. The input provides the depth of the order and corresponding weights ($1/\langle d\rangle^{k-1}$ in this paper) for the aggregating process, as well as the one-hot coding for all nodes. Thus, lines 1-5 of Algorithm 1 generates the initial node embedding h_i^0 based on the weighted matrix $\overline{A}$. Then, lines 6-8 obtain the final node embedding via a multi-channel single-layer neural network, with σ as the activation function. At last, The embedded edge representations are obtained through the proposed bi-directional concatenation in lines 9-13.

4. Experiment and results

4.1. Datasets and metrics

Five benchmark datasets are used in our experiments for the link prediction task, including US-Air [38], Biology [39], Blog [40], Hamster [41], and Yeast [42]. US-Air [38] is an American aviation network graph. The Biology [39] network (Bio-CE-GT) belongs to the category of biological networks. Blog [40] is a U.S. political blog graph. Hamster [41] is a graph of user relationships on the website hamsterster.com. Yeast [42] is an interactive network diagram between yeast

Algorithm 1 Weighted Symmetric Graph Embedding (WSGE) method.

Input: whole graph embedding H^0;

depth K;

adjacency matrices A^k;

activation function σ;

weight δ_k;

one-hot encoding x_i;

nural network f;

concatenating method $g_{forward}$, $g_{backward}$

Output: Edge representations $e_{i,j}$ for $i = 1, 2, \ldots, N; j = 1, 2, \ldots, N; i \neq j$

$\overline{A} = \sum_{k=1}^{K} \delta_k A^k$;

for $i = 1, 2, \ldots, N$ **do**

  $a_i = x_i \overline{A}$;

  $h_i^0 = a_i H^0$;

end for

for $i = 1, 2, \ldots, N$ **do**

  $\overline{h_i} = \sigma(W_1 h_i^0 + b_1) + \sigma(W_2 h_i^0 + b_2)$;

end for

for $i = 1, 2, \ldots, N; j = 1, 2, \ldots, N; i \neq j$ **do**

  $g_{forward}(\overline{h_i}, \overline{h_j}) = cat(\overline{h_i}, \overline{h_j})$; //"cat" means concatenating

  $g_{backward}(\overline{h_i}, \overline{h_j}) = cat(\overline{h_j}, \overline{h_i})$;

  $e_{i,j} = f(g_{forward}(\overline{h_i}, \overline{h_j})) + f(g_{backward}(\overline{h_i}, \overline{h_j}))$;

end for

proteins. The statistics of these datasets are shown in Table 1, $\mid V \mid$ denotes the number of nodes, $\mid E \mid$ refers to the number of edges and d_{avg} represents the average degree. It should be noted that the datasets used in this paper are all undirected, unweighted, and homogeneous networks.

Table 1: Simulation experiment datasets.

Network	$\mid V \mid$	$\mid E \mid$	d_{avg}
US-Air	332	2126	12.81
Biology	924	3239	7.01
Blog	1222	16714	27.36
Hamster	1858	12534	13.49
Yeast	2375	11693	9.85

Three measurements are adopted to evaluate those methods, including the Area Under the receiver operating characteristic Curve (AUC), accuracy(AC), and average precision(AP).

$$accuracy = \frac{TP + TN}{P + N} \tag{6}$$

$$precision = \frac{TP}{TP + FP} \tag{7}$$

AUC curve describes the fraction of true positive rate (TPR) versus the fraction of false positive rate (FPR) at various threshold setting [9]. The true positive rate and false positive rate can be evaluated by the following formulas [11]. True Positive Rate (TPR):

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

False Positive Rate (FPR):

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

To verify the effectiveness of our proposed WSGE approach, we have chose eight graph embedding methods, which are listed as follows.

(1) DeepWalk[22]: DeepWalk is based on the Skip-gram, and employs random walks to obtain the context information (of nodes) for node embedding.

(2) Node2vec[18]: Node2vec further generalizes DeepWalk with Breadth-First Search (BFS) and Depth-First Search (DFS) on random walks.

(3) GCN[33]: GCN is based on the CNN, and is a scalable model since it only aggregates features from local neighborhoods, then characterizes the global neighborhood through multiple iterations.

(4) GraphSAGE[34]: GraphSAGE generates embeddings by sampling the local neighborhood for feature aggregation, and then use a mini-batch forward propagation algorithm to train data.

(5) GAT[35]: GAT introduces the attention mechanism into the propagation process, and assigns different weights to all first-order neighbors.

(6) DeepEdge[30]: In DeepEdge, edges are modeled as functions of nodes, then a new objective-graph likelihood is proposed to jointly optimize the edge function and node representations.

(7) CensNet-VAE[37]: CensNet-VAE is a general graph embedding framework, which embeds both nodes and edges to a latent feature space by using a line graph of the original undirected graph, and combines with Variational Autoencoder.

(8) SEAL[13]: SEAL extracts a local enclosing subgraph around each target link, and then uses GNN to learn general graph structure features for link prediction.

In terms of parameter settings, since DeepWalk and Node2vec are graph embedding methods based on random walks, they share many similar values. For both of them, the length of the walk is 64 and the window size is set to 5. For Node2vec, the return parameter p and the access parameter q are both set to 1. Both GCN and GraphSAGE are graph embedding methods that only consider second-order neighbors. Other comparison methods follow the parameter settings in their original paper.

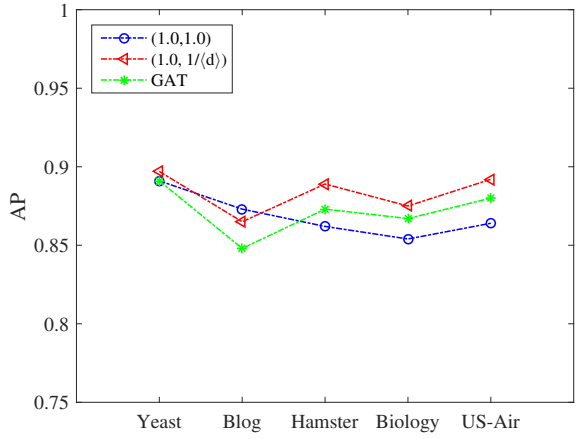
4.2. Evaluation of the proposed weight assignment

WSGE assigns different weights to different orders while the neighbors in the same order share the same weight. The weight of k^{th} order is set as $1/\langle d\rangle^{k-1}$ where $\langle d\rangle$ is the average degree of the corresponding network. Previous methods which involve high-order neighbors only assign the same weight to all neighbors. In contrast, GAT gives all neighbors with different weights while only involves the first order. To compare the performance between different weight assignment methods, we calculate AUC, AC, and AP on all datasets. Figure 6 covers the results when involving neighbors of two orders.

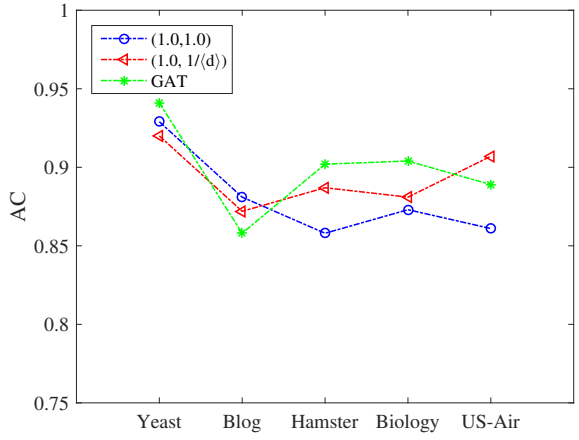
As shown in Figure 6, the WSGE approach has a good performance in almost all datasets, especially referring to AUC. Although the GAT method performs well on some datasets in terms of accuracy, it is the worst in terms of precision and its time complexity is relatively high. Besides, assigning the same weight to all orders only works well on the Blog network. The results indicate that our proposed WSGE approach could well preserve the structural information for link prediction with a fewer time cost.

When involving the third-order neighbors, WSGE assigns the weights for neighbors in these three orders with 1, $1/\langle d\rangle$, and $1/\langle d\rangle^2$, respectively. Results are shown in Figure 7.

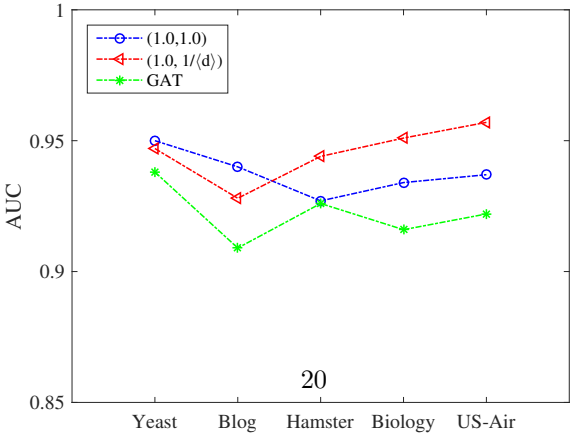
When involving the third-order neighbors, the proposed WSGE approach still achieves the best performance. However, compared to Figure 6, involving the third-order neighbors is not better than considering only the first two orders and may even lead to worse results. The reason is that general complex networks obey the power-law distribution. That is, the number of second-order neighbors is far smaller than that of the third-order. Therefore, involving the third-order neighbors will introduce noise data (false link information), which will drag down the link prediction results. It is in accord with the results of Zhang et al. [13], who have proved that the effective order of node embedding is not that high. In fact, the second-order can safely guarantee to learn high-order features. Thus, our WSGE approach only involves the first-order and second-order neighbors in the following experiments.



(a)

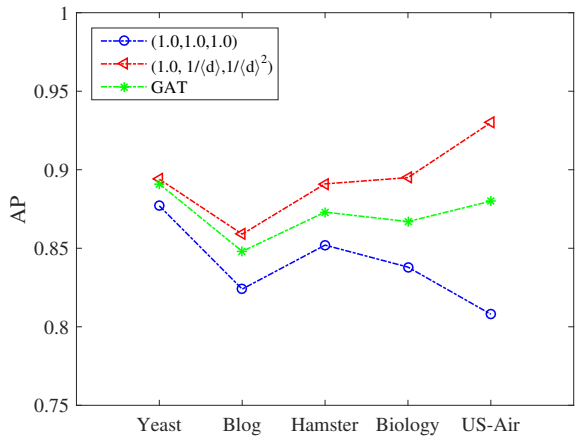


(b)

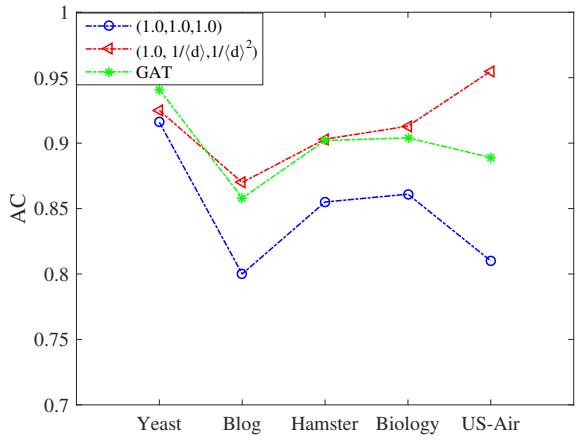


(c)

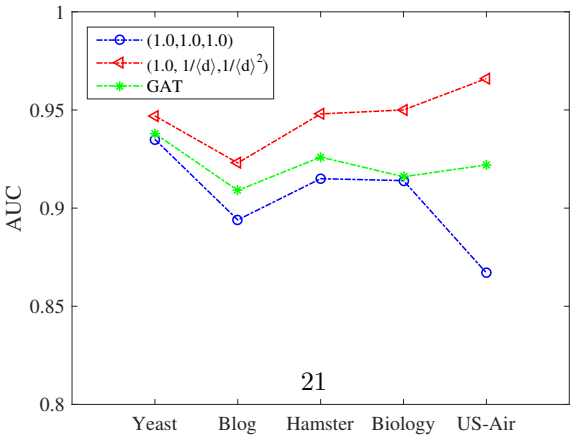
Figure 6: Results when involving neighbors of the first two orders.



(a)



(b)



(c)

Figure 7: Results when involving neighbors of the first three orders.

4.3. Evaluation of the proposed bi-directional concatenating

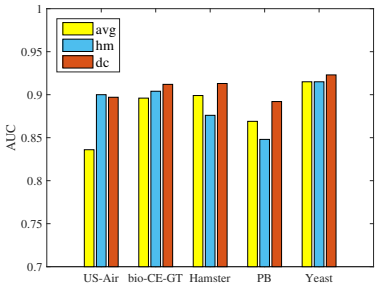
When generating edge embeddings from node representations, a certain concatenating method should be involved, such as mean, Hadamard Product, direct concatenating, and bi-directional concatenating proposed in this paper. In this subsection, we compare the impact of different concatenating methods on the final prediction result. The node embeddings of DeepWalk, Node2vec, GCN, GAT, GraphSAGE, and WSGE are combined with different concatenating methods to demonstrate their corresponding performance on the link prediction task. Results are shown in Figure 8.

In Figure 8, 'avg', 'hm', 'dir' and 'dc' denote employing mean, Hadamard Product, direct concatenating (only applicable for WSGE), and bi-directional concatenating approach proposed in this paper, respectively. It should be noted that only AUC is calculated to compare different concatenating methods w.r.t. their performance on the link prediction task. Through the above experiments, the bi-directional concatenating approach proposed in this paper can be easily combined with various node embedding methods. Compared with other concatenating methods such as mean, Hadamard Product, and direct concatenating, the proposed bi-directional concatenating approach achieves the best performance in almost all cases. It is due to that the bi-directional concatenation can completely preserve the local structural characteristics of the symmetric edge as well as two connected nodes, which results in better performance on the link prediction task.

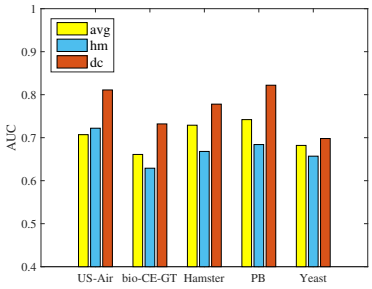
4.4. Performance of link prediction

The performances of different methods on the link prediction task are measured with AUC, AC, and AP. US-Air, Biology, Blog, Hamster, and Yeast are adopted as benchmark datasets.

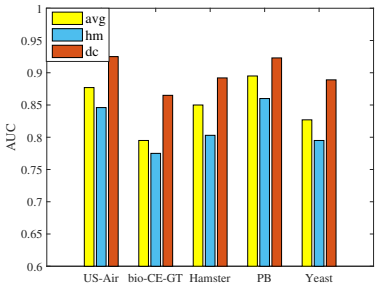
For an undirected, unweighted network, all edges are symmetric, i.e. there is no difference between edge (i, j) and (j, i). For all datasets, we employ the entire adjacency matrix (except the diagonal elements) other than the upper triangular matrix of the adjacency matrix. In other words, if (i, j) appears in



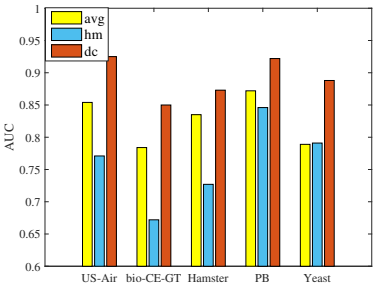
(a) DeepWalk



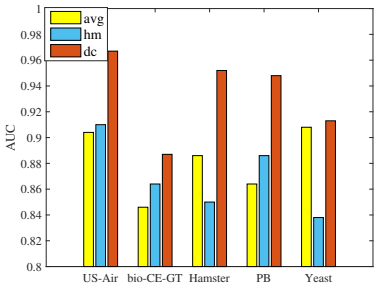
(b) Node2Vec



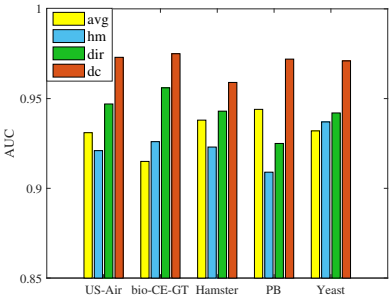
(c) GCN



(d) GAT



(e) GraphSAGE



(f) WSGE

Figure 8: Results of different concatenating methods.

the training set, the testing set may contain (j, i). To test these methods on symmetric data prediction, 60% of data is employed as the training set and 40% of the data is employed as the testing set. The results of our proposed WSGE

and other compared methods are shown in Table 2.

Compared with those methods, our proposed WSGE approach shows better performance as it takes the symmetry of edges into account for undirected graphs. For WSGE, the average values of the three measures AUC, AC, and AP on the five datasets are 0.97, 0.9334, and 0.9252, respectively. It can be seen from Table 2 that WSGE is better than other methods in almost all cases. DeepEdge is better than WSGE w.r.t. AC on the US-Air network and SEAL is better than WSGE w.r.t AP on the Blog network.

Table 2: Results on link prediction.

Network		DeepWalk	Node2vec	GCN	GAT	GraphSAGE	DeepEdge	CensNet	SEAL	WSGE
US-Air	AUC	0.8034	0.8359	0.8530	0.9040	0.9012	0.8920	0.8895	0.9460	**0.9730**
	AC	0.8606	0.7795	0.7950	0.9170	0.8383	**0.9240**	0.9028	0.8703	0.9170
	AP	0.7793	0.7842	0.7769	0.9040	0.8145	0.7570	0.8042	0.8472	**0.9170**
Biology	AUC	0.9114	0.7493	0.7620	0.8610	0.8611	0.8740	0.9137	0.9510	**0.9750**
	AC	0.8992	0.6844	0.7880	0.8630	0.7779	0.8600	0.8333	0.9141	**0.9500**
	AP	0.7932	0.6856	0.7350	0.8610	0.7701	0.7650	0.8343	0.9082	**0.9330**
Blog	AUC	0.8693	0.8145	0.9070	0.9130	0.9114	0.8820	0.9046	0.9313	**0.9590**
	AC	0.8329	0.7437	0.8570	0.8570	0.8300	0.8230	0.8379	0.8436	**0.9020**
	AP	0.7706	0.7430	0.8650	0.8380	0.8427	0.8050	0.8381	**0.9250**	0.9080
Hamster	AUC	0.8962	0.7812	0.8157	0.8500	0.8900	0.8530	0.9081	0.9452	**0.9720**
	AC	0.9158	0.7178	0.7310	0.8220	0.7780	0.8860	0.8357	0.8901	**0.9320**
	AP	0.8034	0.7165	0.7328	0.7690	0.8150	0.6950	0.8359	0.9105	**0.9290**
Yeast	AUC	0.9133	0.7145	0.8340	0.8390	0.9260	0.9160	0.9190	0.9585	**0.9710**
	AC	0.9527	0.6619	0.8260	0.8590	0.8100	0.8910	0.8564	0.8984	**0.9660**
	AP	0.8192	0.6621	0.7930	0.7670	0.8480	0.8300	0.8567	0.8946	**0.9390**

5. Conclusion

Link prediction is an important task in social network analysis and mining because of its numerous applications. Deep learning-based embedding exhibits excellent performance in the task of link prediction. However, there still remain some unsolved problems for this kind of methods. On one hand, they share the same weight to neighbors of different orders during the process of the node

propagating and updating. On the other hand, they can hardly keep the symmetry of embedded edges obtained from node representations by some binary operation methods like mean, Hadamard Product and direct concatenation.

To address these problems, this paper proposes a novel embedding approach WSGE for link prediction. The proposed approach aggregates neighbors in different orders with different weights to obtain node representations. After that, the proposed approach concatenates node pairs bi-directionally (both forwardly and backwardly) and then mixes to guarantee the symmetry of edge representations while preserving local structural information. Experimental results on five datasets show that, through the appropriate weight assignment and bi-directional concatenation, the proposed WSGE can better predict the possibility of existing links in a network, outperforming state-of-the-art methods.

For future steps, we are interested in the following directions. (1) When aggregation, involving all nodes in each order may introduce noise. Therefore, we can further select the appropriate nodes of each order based on their contribution to the central node. (2) When concatenating, the different contributions of nodes at both ends are not considered. In future work, we can assign different weights in line with the contribution of two nodes during the edge formation process.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.61876186, No.61977061, No.61402482) and the Fundamental Research Funds for the Central Universities of China (No.2019XKQYMS85).

References

[1] H. Ghorbanzadeh, A. Sheikhahmadi, M. Jalili, S. Sulaimany, A hybrid method of link prediction in directed graphs, Expert Systems with Applications 165 (2021) 113896.

[2] H. Fan, F. Zhang, Y. Wei, Z. Li, C. Zou, Y. Gao, Q. Dai, Heterogeneous hypergraph variational autoencoder for link prediction, IEEE Transactions on Pattern Analysis and Machine Intelligence (2021) 1–1.

[3] Z. Li, X. Fang, O. R. L. Sheng, A survey of link recommendation for social networks: methods, theoretical foundations, and future research directions, ACM Transactions on Management Information Systems 9 (1) (2017) 1–26.

[4] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proceedings of the IEEE 104 (1) (2015) 11–33.

[5] N. N. Daud, S. H. Ab Hamid, M. Saadoon, F. Sahran, N. B. Anuar, Applications of link prediction in social networks: A review, Journal of Network and Computer Applications 166 (2020) 102716.

[6] H.-M. Cheng, Y.-Z. Ning, Z. Yin, C. Yan, X. Liu, Z.-Y. Zhang, Community detection in complex networks using link prediction, Modern Physics Letters B 32 (01) (2018) 1850004.

[7] Z. Wang, L. Zheng, Y. Li, S. Wang, Linkage based face clustering via graph convolution network, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 1117–1125.

[8] K. G. Quach, P. Nguyen, H. Le, T.-D. Truong, C. N. Duong, M.-T. Tran, K. Luu, Dyglip: A dynamic graph model with link prediction for accurate multi-camera multiple object tracking, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 13784–13793.

[9] S. Haghani, M. R. Keyvanpour, A systemic analysis of link prediction in social network, Artificial Intelligence Review 52 (3) (2019) 1961–1995.

[10] H. Yuliansyah, Z. A. Othman, A. A. Bakar, Taxonomy of link prediction for social network analysis: A review, IEEE Access 8 (2020) 183470–183487.

[11] A. Kumar, S. S. Singh, K. Singh, B. Biswas, Link prediction techniques, applications, and performance: A survey, Physica A: Statistical Mechanics and its Applications 553 (2020) 124289.

[12] A. Divakaran, A. Mohan, Temporal link prediction: A survey, New Generation Computing 38 (1) (2020) 213–258.

[13] M. Zhang, Y. Chen, Link prediction based on graph neural networks, Advances in Neural Information Processing Systems 31 (2018) 5165–5175.

[14] C. Wang, V. Satuluri, S. Parthasarathy, Local probabilistic models for link prediction, in: Seventh IEEE international conference on data mining, 2007, pp. 322–331.

[15] L. Backstrom, J. Leskovec, Supervised random walks: predicting and recommending links in social networks, in: Proceedings of the fourth ACM international conference on Web search and data mining, 2011, pp. 635–644.

[16] H. Wang, X. Shi, D.-Y. Yeung, Relational deep learning: A deep latent variable model for link prediction, in: Thirty-first AAAI conference on artificial intelligence, 2017, pp. 2688–2694.

[17] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, Knowledge-Based Systems 151 (2018) 78–94.

[18] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[19] X. Ma, P. Sun, G. Qin, Nonnegative matrix factorization algorithms for link prediction in temporal networks using graph communicability, Pattern Recognition 71 (2017) 361–374.

[20] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering., in: Nips, Vol. 14, 2001, pp. 585–591.

[21] S. T. Roweis, L. K. Saul, Nonlinear dimensionality reduction by locally linear embedding, science 290 (5500) (2000) 2323–2326.

[22] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.

[23] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 1225–1234.

[24] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, 2016.

[25] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A. J. Smola, Distributed large-scale natural graph factorization, in: Proceedings of the 22nd international conference on World Wide Web, 2013, pp. 37–48.

[26] S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, in: Proceedings of the 24th ACM international on conference on information and knowledge management, 2015, pp. 891–900.

[27] S. Wang, J. Arroyo, J. T. Vogelstein, C. E. Priebe, Joint embedding of graphs, IEEE transactions on pattern analysis and machine intelligence 43 (4) (2021) 1324–1336.

[28] H. Cai, V. W. Zheng, K. C.-C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Transactions on Knowledge and Data Engineering 30 (9) (2018) 1616–1637.

[29] H. Peng, J. Li, H. Yan, Q. Gong, S. Wang, L. Liu, L. Wang, X. Ren, Dynamic network embedding via incremental skip-gram with negative sampling, Science China Information Sciences 63 (10) (2020) 1–19.

[30] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, Learning edge representations via low-rank asymmetric projections, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 1787–1796.

[31] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE transactions on neural networks 20 (1) (2008) 61–80.

[32] Z. Liu, J. Zhou, Introduction to graph neural networks, Synthesis Lectures on Artificial Intelligence and Machine Learning 14 (2) (2020) 1–127.

[33] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907.

[34] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1025–1035.

[35] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018.

[36] C. Wang, C. Wang, Z. Wang, X. Ye, P. S. Yu, Edge2vec: Edge-based social network embedding, ACM Transactions on Knowledge Discovery from Data 14 (4) (2020) 1–24.

[37] X. Jiang, R. Zhu, S. Li, P. Ji, Co-embedding of nodes and edges with graph neural networks, IEEE Transactions on Pattern Analysis and Machine Intelligence (2020) 1–1.

[38] Vladimir batagelj and andrej mrvar (2006): Pajek datasets, `http://vlado.fmf.uni-lj.si/pub/networks/data`.

[39] R. Rossi, N. Ahmed, The network data repository with interactive graph analytics and visualization, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 4292–4293.

[40] R. Ackland, et al., Mapping the us political blogosphere: Are conservative bloggers more prominent?, in: BlogTalk Downunder 2005 Conference, Sydney, 2005.

[41] J. Kunegis, Konect: the koblenz network collection, in: Proceedings of the 22nd international conference on World Wide Web, 2013, pp. 1343–1350.

[42] C. Von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, P. Bork, Comparative assessment of large-scale data sets of protein–protein interactions, Nature 417 (6887) (2002) 399–403.